AFRL-IF-RS-TR-2006-256
Final Technical Report
July 2006

AUTHORITATIVE SSL AUDITOR

Network Resonance, Inc.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

AFRL-IF-RS-TR-2006-256 has been reviewed and is approved for publication

APPROVED: /s/

ROBERT L. KAMINSKI
Project Engineer

FOR THE DIRECTOR: /s/

WARREN H. DEBANY, Jr.
Technical Advisor, Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE	*Form Approved* **OMB No. 0704-0188**

1. REPORT DATE *(DD-MM-YYYY)* JUL 2006	**2. REPORT TYPE** Final	**3. DATES COVERED** *(From - To)* Feb 05 – Jun 06

4. TITLE AND SUBTITLE AUTHORITATIVE SSL AUDITOR	**5a. CONTRACT NUMBER** FA8750-05-C-0295
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER N/A
6. AUTHOR(S) Eric Rescoria and Kevin Dick	**5d. PROJECT NUMBER** DHSF
	5e. TASK NUMBER CL
	5f. WORK UNIT NUMBER AY
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Network Resonance, Inc. 3246 Louis Road Palo Alto, CA 94303-4175	**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFRL/IFGA 525 Brooks Rd Rome NY 13441-4505	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2006-256

12. DISTRIBUTION AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# 06-553

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The Authoritative SSL Auditor provides evidence on demand of any interaction over the network. It is a completely self-contained and absolutely passive device, yielding very low total cost of ownership and very low impact on running applications. It records both unencrypted and encrypted network interactions, then signs them with a US Government certified hardware security module. This process creates extremely strong evidence for unencrypted traffic and practically unimpeachable evidence for encrypted traffic. Because the original network communications is its source, the ASA reflects what actually happened instead of what a particular system thinks happened.

15. SUBJECT TERMS

Cyber Security, Cyber Crime, Homeland Security

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Robert Kaminski
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UL	18	**19b. TELEPONE NUMBER (*Include area code*)**

Table of Contents

1.0 PURPOSE AND ORGANIZATION OF THIS REPORT

The purpose of this report is to review the lessons learned during the development of the Authoritative SSL Auditor under contract FA8750-05-C-0295. To illustrate these lessons, it extensively references the Final System Design Documentation delivered during the Final Development phase of the project. To assist the reader by providing a general context for the discussion of lessons learned, this report reproduces the first two sections of that documentation as *Section 2.0 Background* and *Section 3.0 System Overview*. However, to understand the technical details underlying specific lessons, the reader should have access to a complete copy of the Final System Design Documentation.

After *Section 2.0 Background* and *Section 3.0 System Overview*, *Section 4.0 Development Process Overview* summarizes how we organized the project tasks. Obviously, the process we used significantly affected our learning, so reviewing it is another important part of setting the stage. Finally, *Section 4.0 Lessons Learned* presents several key areas of technical insight. We hope that these insights prove valuable to other researchers on future projects.

2.0 BACKGROUND

The Network Resonance Authoritative SSL Auditor (ASA) enables commercial enterprises and government agencies to produce evidence on demand for electronic transactions with zero drag on operational systems and zero delay in producing proof. Such organizations are subject to a bewildering array of regulations and policies. Figuring out the corresponding information technology (IT) requirements for compliance presents a challenge. Overlapping regulations, evolving interpretations, and revised practices create a poorly visible, constantly moving target. However, there is one consistent requirement for a sound defense—evidence. In all cases, compliance requires the ability to provide corresponding proof.

There are two considerations: cost and quality. With all the other potential costs of updating systems to comply with policies, organizations want to minimize the necessary investment in evidence infrastructure. At the same time, they don't want to sacrifice quality by relying on evidentiary measures that won't withstand serious scrutiny. They also don't want to compromise other compliance objectives, like confidentiality, or operational objectives, like service levels.

The Authoritative SSL Auditor provides evidence on demand of any interaction over the network. It is a completely self-contained and absolutely passive device, yielding very low total cost of ownership and very low impact on running applications. It records both unencrypted and encrypted network interactions, then signs them with a U.S. government certified hardware security module. This process creates extremely strong evidence for unencrypted traffic and practically unimpeachable evidence for encrypted traffic. Because the original network communications is its source, the ASA reflects what actually happened instead of what a particular system thinks happened.

Reviewing the evidence is simple. A Web interface enables authorized personnel to retrieve any recording and replay it. For Web interactions, they view the sequence of pages. For voice interactions, they listen to the conversation. For messaging interactions, they read the transcript. Because all recordings include low-level networking data as well as the content, they can also see exactly what security measures were applied. For encrypted interactions, a highly specialized protocol ensures that the system replays only the targeted interaction. There is no collateral breach of confidentiality. Private encryption keys never leave the original application.

The most important benefit of the ASA is complete insulation from changes to application software or compliance policies. It replays the network interaction and guarantees the integrity of the recording. If an application's behavior or a regulation's interpretation changes, it will still replay and guarantee. There's no lag in producing evidence. Whatever an official needs to see, the ASA makes it easy to comply.

3.0 SYSTEM OVERVIEW

To minimize barriers to deployment, the most attractive means for generating evidence of network communications is a passive network device as shown in Figure 1. A fabric of routers and switches already connect client and server machines that generate network traffic. Typically, server machines are concentrated in departmental and enterprise data centers, so the switches in these data centers present a convenient opportunity to access this traffic.

Most enterprise-class switches provide a feature called "port mirroring" where the switch forwards a copy of the packets from selected ports to a designated mirror port. In cases where this features is unavailable or inconvenient, a device called a "network tap" can add the same ability as a standalone device. Once a mirror port is configured or a network tap is deployed, creating evidence of the traffic flowing to and from a given switch is simply a matter of plugging in and configuring one of the ASA's Auditing Sensors.

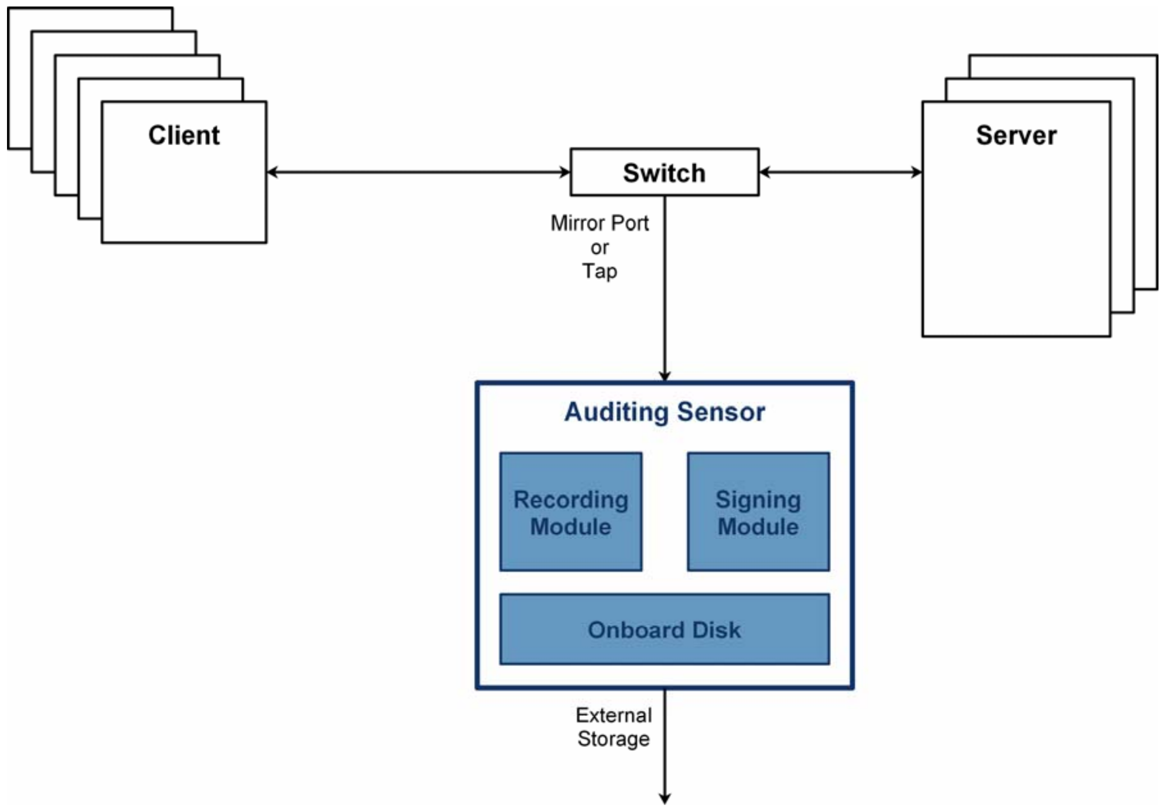

Figure 1 – Recording Topology

As the switch processes packets flowing between the client and server, the Auditing Sensor receives copies via the mirror port or network tap. The Recording Module captures packets flows, identifies coherent communications, and saves them to the Onboard Disk. It also sends these packet sequences to the Signing Module, which then uses a hardware security module (HSM) to cryptographically sign these sequences. Because the HSM is a Federal Information Processing Standards (FIPS) 140 Level 3 or 4 security processor, its cryptographic keys and signature processing are highly resistant to compromise. The combination of the original SSL encryption and HSM signature creates practically unimpeachable evidence of the original communication. The Sensor provides facilities for moving recordings from the Onboard Disk to remote network volumes.

Recording is the first half of the overall auditing process. Replay is the second. As shown in Figure 2, replaying previously recorded traffic requires two additional components. Obviously, viewing a recorded communication requires some sort of user interface. The ASA provides this interface as an Auditing Portal implemented in Java.

For small installations, the Auditing Sensor itself can include the Java portal software that executes the Auditing Portal, yielding a completely self-contained package that is extremely easy to deploy. This approach works well when the number of sensors is relatively low and the volume of traffic they record is modest. For larger installations, the preferred means of deployment is to deploy a separate, dedicated device running the Auditing Portal on top of the necessary Java portal software.

If desired, adopting organizations can integrate the Auditing Portal with Web interfaces for other security or auditing solutions using one of two methods. First, they can use commercial portal software that remotely incorporates individual portals within a larger interface, by "clipping" them. Second, they can use standard Java tools to package the Auditing Portal as Java Specification Request (JSR) 168 portlet and deploy it along with similar portlets as part of a single Java portal instance. Because the Portal uses the most widely accepted Java Web portal frameworks, this integrations is relatively straightforward. The range of Portal deployment options enables the ASA to accommodate installations of varying sizes and requirements.

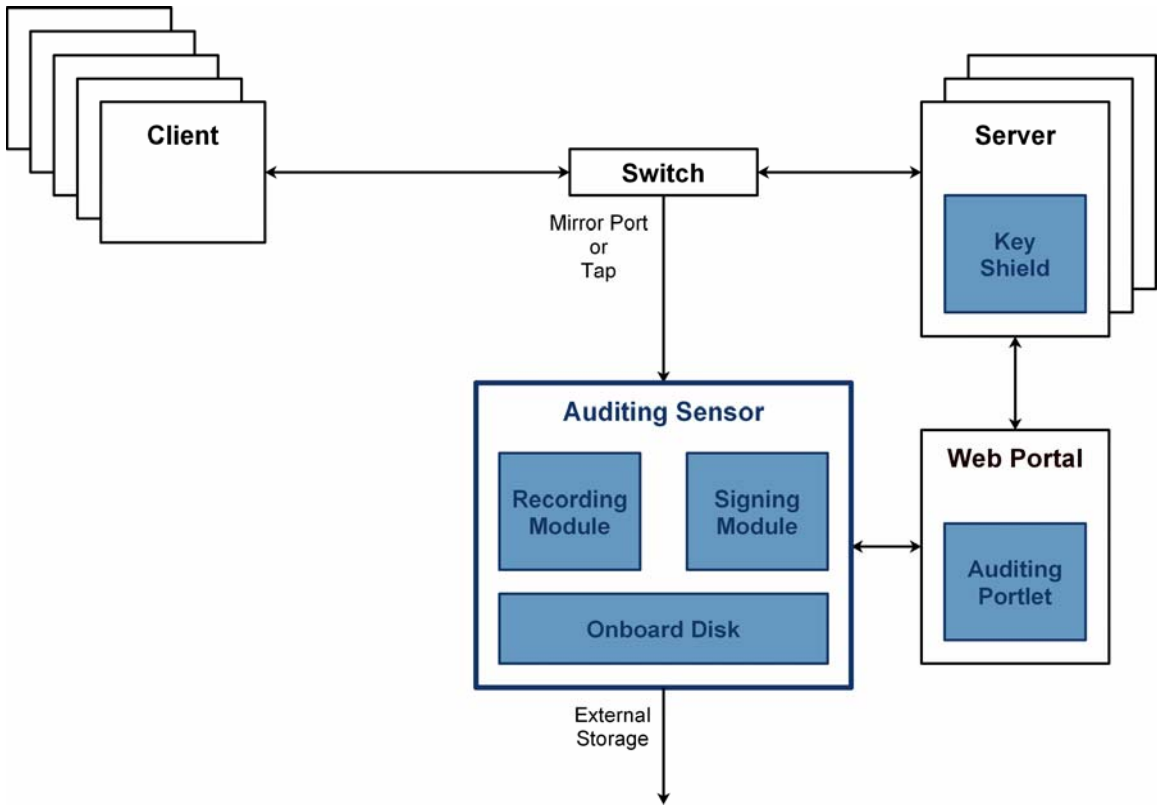

Figure 2 – Replay Topology

The other component required for replay is the Key Shield. All current approaches to examining passively recorded SSL data require that the adopting organization make copies of the private keys used by SSL servers and then distribute them to either the recording devices themselves or machines running viewing software. Because the privacy of these keys is what guarantees that no machine can impersonate a legitimate server, this key distribution degrades organizations' confidentiality measures. Moreover, this distribution substantially erodes the trustworthiness of the audit trails themselves because it raises the question of whether someone falsely generated the trails using copies of private keys.

The ASA overcomes this challenge by performing all operations that involve a server's private keys using the Key Shield, which runs only on that particular server machine. Therefore, a server's keys never leave its physical control.

With these three major components in place, the ASA can provide complete and verifiable evidence of all recorded SSL communications. When an authorized user of the Auditing Portal wants to replay a communication, he first authenticates himself to the Java Web portal. Then he searches for communications that meet his specified criteria. Upon selecting a communication for replay, all the ASA components cooperate to guarantee the integrity of the replay process:

1. The Auditing Portal examines the index of packet flows to determine the location of all the packets constituting a given communication.
2. The Auditing Portal retrieves the identified flows and the corresponding signatures.
3. The Auditing Portal verifies the signatures to ensure that they were made by a legitimate Auditing Sensor and that nobody has since tampered with the flows.
4. The Auditing Portal examines the recording to determine which server machine participated in the communication, copies the part of the recording containing the SSL handshake, and sends it to the Key Shield running on that server.
5. After checking that the Auditing Portal is on its list of authorized Portals, the Key Shield retrieves the appropriate private key from its encrypted file and uses it to decode the necessary SSL handshakes. It extracts the SSL session keys that are specific to that particular communication, encrypts them under the Auditing Sensor's public key, and returns them to the Portal.
6. The Portal then sends the encrypted session keys to the Auditing Sensor for decryption, along with the username of the user who requested the replay.
7. After checking that the Auditing Portal is on its list of authorized Portal, the Sensor creates a log entry that identifies the communication being replayed, the Portal conducting the replay, and the user who requested the replay. It then cryptographically signs this entry and saves it to the audit log. Finally, it decrypts the session keys and returns them to the Portal.
8. The Auditing Portal uses the session keys to decrypt the communication and then reconstructs all the application-level messages exchanged during to communication.
9. The Auditing Portal lets the user navigate through the application-level messages to replay the communication.

While this process is complicated, it is completely hidden from the end-user and provides a high degree of confidence in the resulting evidence. Replaying a secure communication requires the full cooperation of all three major components of the ASA. All replays are logged in the HSM's highly secure storage so nobody can secretly access recordings. Each replay only discloses the contents of a specifically targeted communication.

4.0 DEVELOPMENT PROCESS OVERVIEW

The project was structured in four phases. The first phase, System Analysis, resulted in plans and prototypes. The second phase, Alpha Development, resulted in alpha quality software. The third phase, Beta Development, resulted in beta quality software. The fourth, Final Development, resulted in production grade software.

Spiral development served as the underlying model for this structure. We made several passes at developing the desired system, refining and enhancing at each step. This approach required a substantial amount of organizational agility due to the compounding of standard dependencies among subsystems with dependencies among generations of subsystems from different passes of the spiral.

As we got further into the spiral, different subsystems had progressed at different rates, causing dependency issues. For example, some subsystems progressed rapidly, but had dependencies on other subsystems that progressed slowly. So the rapidly progressing ones would lie fallow for a while because they were ahead of schedule and we needed to concentrate on the slowly progressing ones. Once all subsystems were at the same level of refinement, the dependencies would be broken because the rapidly progressing ones relied on the very early interfaces of the slowly progressing ones. Conversely, when slowly progressing subsystems had dependencies on rapidly progressing ones, multiple evolutions of the interfaces for rapidly progressing subsystems could slow development on the slowly progressing ones even further. In addition to enhancing the functionality of the slowly progressing subsystems, we also had to constantly update them to use the latest interfaces for the rapidly progressing ones. These dependency issues required constant attention.

Luckily, our team was very small and most of its members had worked together for many years. Therefore, the benefits of the spiral approach outweighed the costs. However, we would not recommend a spiral process for larger teams or ones where members were unfamiliar with each other.

5.0 LESSONS LEARNED

5.1 Real World Network Data

One of the primary goals of the project was for the resulting implementation of the Authoritative SSL Auditor (ASA) to be production ready implementation. Because the ASA relies on passively capturing and subsequently reassembling network traffic, one of the primary obstacles to production readiness was access to real world network data. While all of the network protocols leveraged by the ASA are standards defined by the Internet Engineering Task Force (IETF), the reality of modern networks is that they are complicated enough that the standards documents themselves do not provide sufficient information about network behavior.

Building a complete replica of a modern enterprise network would be extremely costly. Luckily, there are standard mechanisms for capturing traces of actual network traffic. However, most traffic contains sensitive data so enterprises are unwilling to share such traces. As discussed in *Section 7.1 Functional Testing* of the Final System Design Documentation, we addressed this issue in two ways.

First, we did manage to collect traces from three commercial Web sites, in two cases based on personal relationships with the site operator and in another based on a commercial relationship. Second, we set up an SSL proxy on our own network and captured interactions of our own engineers with external Web sites such as Amazon. While much better than no real world testing, these samples did not have the breadth we would have preferred in a larger and more diverse sample. In particular, researchers at large organizations could probably generate substantial test traffic using the SSL proxy approach.

Luckily, the decoding and reassembly modules of the ASA are nearly identical to those used in our commercial Passive Capture Engine (PCE) product. We sell this package to vendors of application monitoring software that want to use passive network capture. Through these vendors, the PCE is running on between 50 and 100 large commercial networks. For confidentiality reasons, we don't generally have large traces from those networks, but we do get bug reports and sample TCP connections if the PCE fails to properly reassemble and decode their traffic. Therefore, the ASA has undergone an enormous amount of indirect testing on real world networks.

Of course, most other researchers won't have this advantage and direct testing is clearly preferable to indirect. The PREDICT project (http://www.predict.org), sponsored by the Department of Homeland Security (DHS) Science and Technology (S&T) , is gathering samples of real network traffic. When we needed test traffic, PREDICT did not have a library of full packet traces. We understand such a library is planned for the future. We strongly endorse its creation and recommend that future researchers in this area take advantage of it.

5.2 Commercial Test Partner

In addition to testing with real world traffic, fulfilling the project goals also required the ultimate real world test—running on a production network at a commercial enterprise. We felt this requirement was so important that we originally obtained the agreement of such a partner prior to submitting a research proposal for consideration. Unfortunately, the business environment evolved faster than proposal evaluation and contract approval. Shortly after we finalized the contract, our commercial partner made a business decision to outsource its network operations making it impossible for them to fulfill their promise to help test the system.

As described in *Section 7.31 Field Testing* of the Final System Design Documentation, we did finally identify a replacement commercial test partner. However, the search required a tremendous level of effort and actually delayed project completion. Moreover, because we located this replacement partner very late in the project, we did not achieve the expected benefits of having the partner provide guidance and feedback throughout the lifecycle.

Our assessment is that the primary obstacles to finding this partner were (1) ability to identify decision makers at candidate organizations, (2) availability of qualified personnel to assist in the testing, and (3) concerns over connecting unproven devices to the network. We feel strongly that the Department of Homeland Security Science and Technology Directorate could sponsor a matchmaking program that would help researchers overcome all three of these obstacles. It would be analogous to the PREDICT project discussed above, but for *in situ* testing rather test data.

Key elements of such a program would include commercial organizations registering their interest in specific areas of cybersecurity technology and their availability for assisting with testing. The program operator would serve as a trusted third party, ensuring that research projects had achieve a suitable level of functionality and stability, then connecting them to the commercial organizations that have registered interest in the relevant technology area and are available in the necessary timeframe.

Commercial organizations would benefit because they would achieve visibility into the cybersecurity technology pipeline and guide each technology's evolution to optimize its applicability to their businesses. Obviously, researchers would benefit because they would receive feedback earlier in the process and hopefully more of it. Funding agencies would also benefit by having much better data on the commercial potential of the technologies they fund. With proper metrics in place, the entire process would be amenable to continuous refinement and optimization.

5.3 Java Web Stack Productivity

As described in *Section 4.0 Component Detail: Auditing Portal* of the Final System Design Documentation, the ASA's Auditing Portal relies on the standard Java Web stack to provide the infrastructure for processing Web requests and generating Web pages. The popular characterization of this stack is that it provides a highly productive and agile platform for Web development. This was not our experience.

The first problem we encountered was one of expertise. We were under the impression that the programming skills necessary to use the standard Web Java stack were a readily available commodity. However, we had great difficulty hiring people with these skills and the ones we did hire weren't very capable. We initially hired a Java specialist that was unable to produce any significant functionality. We then hired a different Java specialist that was able to produce such functionality, but only very slowly. Finally, we hired a third programmer who was skilled in Java, though not a specialist. He was able to work productively, in our opinion, because his broader range of skills enabled him to better understand the role of the Auditing Portal within the context of the entire ASA.

As described in *Section 4.0 Component Detail: Auditing Portal*, the standard Java Web uses a suite of components to supposedly simplify development and make applications easier to change. The Java Servlet API provides an execution container for the Web-related components. The Apache Struts Framework simplifies the process of building Web pages that interact with server-side code by providing a declarative dispatch language for invoking different types of Java functionality. The Tiles Framework plugs into the Struts framework and provides a high level grammar for describing Web form interaction.

Our experience was that it took a tremendous amount of time to get an initial version of the Auditing Portal up and running. So the initial pass at the spiral development approach took much longer for the Auditing Portal than any other subsystem, with the knock-on dependency effects described previously. Our opinion is that all of the indirection inherent in the Servlet-Struts-Tiles stack makes it very difficult to transform a conceptualization of the system into the appropriate declarative instructions. However, once we reached critical mass, refinements proceeded incredibly rapidly. Subsequent passes at the spiral actually went faster for the Auditing Portal than the other subsystems, allowing us to make up most of the ground we had lost.

If we had to go back and make the choice again, we would still choose Java over alternatives like PHP. In fact, we were able to do a proof-of-concept port of the Management UI written in PHP in a few days using our Tiles template. However, we would schedule and manage the project differently to account for long ramp up and rapid enhancement times for the standard Java Web stack.

5.4 Working with HSMs

As described in *Section 3.6 Signer* of the Final System Design Documentation, we use an nCipher nShield 150 hardware security module (HSM) to provide authenticity guarantees and a replay audit trail. Implementing this functionality required writing a custom program that executed on the HSM's processor. In a prototype version of the ASA built under a DARPA contract, we used the IBM 4758 HSM. Therefore, we have two data points for working with HSMs.

In both cases, we discovered that writing custom programs was quite difficult, though more so for the nCipher than for the 4758. Unfortunately, the 4758 become commercially unavailable for this project. The problem seems to be that the ability for customers to write custom programs is not the primary purpose of the products. Their primary purpose is to execute applications provided by their respective manufacturers. The internal application developers, of course, have access to the developers that designed the HSM and don't have to rely solely on the published code samples and documentation.

For the nCipher, we found these code samples and documentation to be both a bit sparse and a bit out-of-date. The fact that support personnel seemed unfamiliar with the custom application development process compounded the mediocre quality of the code samples and documentation. Every interaction required assistance from a product engineer, brokered through the support representative. So roundtrips could consume several days. Even worse, limited time prevented product engineers from providing complete answers so resolving an issue typically required many roundtrips.

One of the nCipher's unique features is the sophistication of its permission scheme for controlling access to resources available through the card. *Section 3.6 Signer* describes the scheme we used, which, while complicated, was essentially the simplest possible. While this permission scheme has attractive properties from a security perspective, the documentation provided no guidance in how to put the low level permissions features together into a coherent scheme. We required several consultations with nCipher product engineers and access to internal code that we could use as a model.

Because the HSM is a central feature of the ASA, these difficulties could have proved catastrophic for the project. Luckily, we knew from our experience with the 4758 used on the DARPA project that writing a simulator for the HSM would prove a sound time investment. This simulator consisted of an abstract API and a software only implementation of the API functions. Using the nShield card itself required simply writing a bridge from the abstract API to the car'd native API.

Therefore, while we were waiting for nCipher's assistance, we were still able to proceed with development on the other subsystems. They simply used the simulator. In the case of the Auditing Portal, this model proved invaluable because it depended on the HSM. The trouble getting basic functionality implemented for both subsystems would have been disastrous if it weren't for the isolation provided by the simulator.

5.5 Dynamic Decoder Chains

As described in *Section 2.0 System Overview* of the Final System Design Documentation, we designed the sections of the recording and replay system that reconstructed protocol messages as a stack of dynamic decoder chains. This approach stemmed from our early experience on the DARPA project and building the PCE. The DARPA software used a mostly monolithic decoding stack. The PCE software used a modular, but static decoding stack.

For this project, we completed the evolution to modular, dynamic decoding chains. The Recorder and Player Core can assemble these chains from dynamically loadable shared objects based on the directives contained in the Registry. Obviously, this approach will make it much easier to extend the ASA to support auditing of applications other than the Web.

It also has to less obvious benefits. First, extending the system in small ways will be much easier too. If we need to process the data stream and add metadata that will assist in later retrieval, we can implement this functionality as a new decoder, greatly simplifying the development and testing process.

Second, testing the system was much easier because we were able to create a decoder chain specifically for automated testing as discussed in *Section 7.1 Functional Testing* of the Final System Design Documentation. So instead of having to test replay functionality in a two step process of recording and replay, we were able to create a combined recording and replay stack. This stack tested the end-to-end ability of the system to accurately reconstruct the original messages in a single, automated pass.

5.6 Database Performance

As described in *3.4 Indexer and Session Index* of the Final System Design Documentation, we tried using both Berkeley DB and PostgreSQL to manage the Session Index. We actually evaluated even more solutions, including other open source relational databases and object databases. Figuring out how to manage the Session Index turned out to be much more time consuming than we originally envisioned and at times we thought it might not be feasible to simultaneously achieve the write and read performance we desired.

The fundamental problem is that all common database managers are optimized to provide fast reads for concurrent users, with a certain number of writes. In contrast, the usage pattern for the ASA is to constantly write an enormous amount of data but to only occasionally read select, small parts of it. Being able to search through the large volumes of data to find select parts requires indexing.

Unfortunately, most database indexing schemes make updating them very expensive. The usage model they assume is that writes are infrequent enough that they need to update the index for each individual write and they need to update the entire index. The ASA can potentially write thousands of records per second and the typical indexing scheme can quickly become the dominant performance cost.

The two-pronged strategy of using copy operations instead of insert operations and partitioning the index as described in *3.4 Indexer and Session Index* overcame this issue with PostgreSQL. The partitioning feature is new in version 8.1 and many researchers may not be aware of it. Moreover, it wasn't obvious how to leverage this feature to sole the indexing cost problem and required a fair amount of experimentation. Hopefully, our experience can help other researchers avoid this effort.

5.7 Disk I/O Performance

While the PostgreSQL optimizations discussed in the previous section enabled us to achieve the project's performance targets, we did not have as much headroom as we would have liked. The bottleneck appears to be disk I/O performance. We suspect a fundamental limitation of common SATA controllers.

Prior to specifying the device hardware for this project, we did extensive research on disk performance. We knew that we would be streaming data to disk at a rapid rate and wanted to be sure the I/O subsystem could handle it. All the testing and benchmarks we reviewed suggested that a SATA RAID system would be sufficient.

We originally configured the disks on the device in a RAID 0 array, where the controller card stripes data across multiple disks. This approach theoretically speeds both reads and writes. Each logical operation is divided into multiple parallel operations across the available disks.

However, we experienced dramatic decreases in throughput when we tried to read and write simultaneously to the same RAID volume. We suspect this behavior may be an artifact of the RAID controller chips, but we aren't sure. In any case, we had to turn RAID off and configure file locations to minimize simultaneous reads and writes to the same disk.

Even then, it was clear that we would not be able to exceed our performance targets by much due to the I/O bottleneck. Because we did meet these targets, we did not to any further experimentation. The first thing we would try is the SCSI interface rather than the SATA interface. The next thing we would try is using 15,000rpm disks rather than 10,000rpm disk. While 15,000rpm SCSI disks are substantially more expensive than 10,000rpm SATA disks, we recommend that researchers working on systems with simultaneous read/write access to the same disks under demanding performance requirements make the investment because it will likely make up the difference by decreasing the necessary amount of performance tuning experimentation.